msgpack Documentation

Release 1.0

Author

2020-12-22

Contents

MessagePack is a efficient format for inter language data exchange.

CHAPTER 1

API reference

`msgpack.pack(o, stream, **kwargs)`

Pack object *o* and write it to *stream*

See `Packer` for options.

`dump()` is alias for `pack()`

`msgpack.packb(o, **kwargs)`

Pack object *o* and return packed bytes

See `Packer` for options.

`dumps()` is alias for `packb()`

`msgpack.unpack(stream, **kwargs)`

Unpack an object from *stream*.

Raises *ExtraData* when *stream* contains extra bytes. See `Unpacker` for options.

`load()` is alias for `unpack()`

`msgpack.unpackb(packed, *, object_hook=None, list_hook=None, bool use_list=True, bool raw=False, int timestamp=0, bool strict_map_key=True, unicode_errors=None, object_pairs_hook=None, ext_hook=ExtType, Py_ssize_t max_str_len=-1, Py_ssize_t max_bin_len=-1, Py_ssize_t max_array_len=-1, Py_ssize_t max_map_len=-1, Py_ssize_t max_ext_len=-1)`

Unpack packed_bytes to object. Returns an unpacked object.

Raises `ExtraData` when *packed* contains extra bytes. Raises `ValueError` when *packed* is incomplete. Raises `FormatError` when *packed* is not valid msgpack. Raises `StackError` when *packed* contains too nested. Other exceptions can be raised during unpacking.

See `Unpacker` for options.

max_xxx_len options are configured automatically from `len(packed)`.

`loads()` is alias for `unpackb()`

class msgpack.**Packer**(*default=None*, ***, *bool use_single_float=False*, *bool autoreset=True*, *bool use_bin_type=True*, *bool strict_types=False*, *bool datetime=False*, *unicode_errors=None*)

MessagePack Packer

Usage:

```
packer = Packer()
astream.write(packer.pack(a))
astream.write(packer.pack(b))
```

Packer's constructor has some keyword arguments:

Parameters

- **default** (`callable`) – Convert user type to builtin type that Packer supports. See also simplejson's document.
- **use_single_float** (`bool`) – Use single precision float type for float. (default: False)
- **autoreset** (`bool`) – Reset buffer after each pack and return its content as *bytes*. (default: True). If set this to false, use *bytes()* to get content and *.reset()* to clear buffer.
- **use_bin_type** (`bool`) – Use bin type introduced in msgpack spec 2.0 for bytes. It also enables str8 type for unicode. (default: True)
- **strict_types** (`bool`) – If set to true, types will be checked to be exact. Derived classes from serializeable types will not be serialized and will be treated as unsupported type and forwarded to default. Additionally tuples will not be serialized as lists. This is useful when trying to implement accurate serialization for python types.
- **datetime** (`bool`) – If set to true, datetime with tzinfo is packed into Timestamp type. Note that the tzinfo is stripped in the timestamp. You can get UTC datetime with *timestamp=3* option of the Unpacker. (Python 2 is not supported).
- **unicode_errors** (`str`) – The error handler for encoding unicode. (default: 'strict') DO NOT USE THIS!! This option is kept for very specific usage.

bytes(*self*)

Return internal buffer contents as bytes object

getbuffer(*self*)

Return view of internal buffer.

pack(*self*, *obj*)

pack_array_header(*self*, *long long size*)

pack_ext_type(*self*, *typecode*, *data*)

pack_map_header(*self*, *long long size*)

pack_map_pairs(*self*, *pairs*)

Pack *pairs* as msgpack map type.

pairs should be a sequence of pairs. (*len(pairs)* and *for k, v in pairs:* should be supported.)

reset(*self*)

Reset internal buffer.

This method is useful only when autoreset=False.

class msgpack.**Unpacker**(*file_like=None, Py_ssize_t read_size=0, *, bool use_list=True, bool raw=False, int timestamp=0, bool strict_map_key=True, object_hook=None, object_pairs_hook=None, list_hook=None, unicode_errors=None, Py_ssize_t max_buffer_size=104857600, ext_hook=ExtType, Py_ssize_t max_str_len=-1, Py_ssize_t max_bin_len=-1, Py_ssize_t max_array_len=-1, Py_ssize_t max_map_len=-1, Py_ssize_t max_ext_len=-1*)

MessagePack Packer

Usage:

```
packer = Packer()
astream.write(packer.pack(a))
astream.write(packer.pack(b))
```

Packer's constructor has some keyword arguments:

Parameters

- **default** (*callable*) – Convert user type to builtin type that Packer supports. See also simplejson's document.
- **use_single_float** (*bool*) – Use single precision float type for float. (default: False)
- **autoreset** (*bool*) – Reset buffer after each pack and return its content as *bytes*. (default: True). If set this to false, use *bytes()* to get content and *.reset()* to clear buffer.
- **use_bin_type** (*bool*) – Use bin type introduced in msgpack spec 2.0 for bytes. It also enables str8 type for unicode. (default: True)
- **strict_types** (*bool*) – If set to true, types will be checked to be exact. Derived classes from serializable types will not be serialized and will be treated as unsupported type and forwarded to default. Additionally tuples will not be serialized as lists. This is useful when trying to implement accurate serialization for python types.
- **datetime** (*bool*) – If set to true, datetime with tzinfo is packed into Timestamp type. Note that the tzinfo is stripped in the timestamp. You can get UTC datetime with *timestamp=3* option of the Unpacker. (Python 2 is not supported).
- **unicode_errors** (*str*) – The error handler for encoding unicode. (default: 'strict') DO NOT USE THIS!! This option is kept for very specific usage.

Example of streaming deserialize from file-like object:

```
unpacker = Unpacker(file_like)
for o in unpacker:
    process(o)
```

Example of streaming deserialize from socket:

```
unpacker = Unpacker()
while True:
    buf = sock.recv(1024**2)
    if not buf:
        break
    unpacker.feed(buf)
    for o in unpacker:
        process(o)
```

Raises `ExtraData` when *packed* contains extra bytes. Raises `OutOfData` when *packed* is incomplete. Raises `FormatError` when *packed* is not valid msgpack. Raises `StackError` when *packed* contains too nested. Other exceptions can be raised during unpacking.

`feed`(*self*, *next_bytes*)
: Append *next_bytes* to internal buffer.

`read_array_header`(*self*)
: assuming the next object is an array, return its size n, such that the next n unpack() calls will iterate over its contents.

  Raises *OutOfData* when there are no more bytes to unpack.

`read_bytes`(*self*, *Py_ssize_t nbytes*)
: Read a specified number of raw bytes from the stream

`read_map_header`(*self*)
: assuming the next object is a map, return its size n, such that the next n * 2 unpack() calls will iterate over its key-value pairs.

  Raises *OutOfData* when there are no more bytes to unpack.

`skip`(*self*)
: Read and ignore one object, returning None

  Raises *OutOfData* when there are no more bytes to unpack.

`tell`(*self*)
: Returns the current position of the Unpacker in bytes, i.e., the number of bytes that were read from the input, also the starting position of the next object.

`unpack`(*self*)
: Unpack one object

  Raises *OutOfData* when there are no more bytes to unpack.

`class` `msgpack.`**`ExtType`**
: ExtType represents ext type in msgpack.

`class` `msgpack.`**`Timestamp`**(*seconds*, *nanoseconds=0*)
: Timestamp represents the Timestamp extension type in msgpack.

  When built with Cython, msgpack uses C methods to pack and unpack *Timestamp*. When using pure-Python msgpack, `to_bytes()` and `from_bytes()` are used to pack and unpack *Timestamp*.

  This class is immutable: Do not override seconds and nanoseconds.

  `__init__`(*seconds*, *nanoseconds=0*)
  : Initialize a Timestamp object.

    Parameters

    - **seconds** (*int*) – Number of seconds since the UNIX epoch (00:00:00 UTC Jan 1 1970, minus leap seconds). May be negative.
    - **nanoseconds** (*int*) – Number of nanoseconds to add to *seconds* to get fractional time. Maximum is 999_999_999. Default is 0.

    Note: Negative times (before the UNIX epoch) are represented as negative seconds + positive ns.

  `static from_bytes`(*b*)
  : Unpack bytes into a *Timestamp* object.

    Used for pure-Python msgpack unpacking.

Parameters **b** (*bytes*) – Payload from msgpack ext message with code -1

Returns Timestamp object unpacked from msgpack ext payload

Return type *Timestamp*

static from_datetime(*dt*)
Create a Timestamp from datetime with tzinfo.

Python 2 is not supported.

Return type *Timestamp*

static from_unix(*unix_sec*)
Create a Timestamp from posix timestamp in seconds.

Parameters **unix_float** (*int or float.*) – Posix timestamp in seconds.

static from_unix_nano(*unix_ns*)
Create a Timestamp from posix timestamp in nanoseconds.

Parameters **unix_ns** (*int*) – Posix timestamp in nanoseconds.

Return type *Timestamp*

to_bytes()
Pack this Timestamp object into bytes.

Used for pure-Python msgpack packing.

Returns data Payload for EXT message with code -1 (timestamp type)

Return type bytes

to_datetime()
Get the timestamp as a UTC datetime.

Python 2 is not supported.

Return type datetime.

to_unix()
Get the timestamp as a floating-point value.

Returns posix timestamp

Return type float

to_unix_nano()
Get the timestamp as a unixtime in nanoseconds.

Returns posix timestamp in nanoseconds

Return type int

1.1 exceptions

These exceptions are accessible via *msgpack* package. (For example, *msgpack.OutOfData* is shortcut for *msgpack.exceptions.OutOfData*)

exception msgpack.exceptions.**BufferFull**
Bases: *msgpack.exceptions.UnpackException*

exception `msgpack.exceptions.`**`ExtraData`**(*unpacked*, *extra*)

Bases: `ValueError`

ExtraData is raised when there is trailing data.

This exception is raised while only one-shot (not streaming) unpack.

exception `msgpack.exceptions.`**`FormatError`**

Bases: `ValueError`, *`msgpack.exceptions.UnpackException`*

Invalid msgpack format

exception `msgpack.exceptions.`**`OutOfData`**

Bases: *`msgpack.exceptions.UnpackException`*

exception `msgpack.exceptions.`**`StackError`**

Bases: `ValueError`, *`msgpack.exceptions.UnpackException`*

Too nested

exception `msgpack.exceptions.`**`UnpackException`**

Bases: `Exception`

Base class for some exceptions raised while unpacking.

NOTE: unpack may raise exception other than subclass of UnpackException. If you want to catch all error, catch Exception instead.

CHAPTER 2

Advanced usage

2.1 Packer

2.1.1 autoreset

When you used `autoreset=False` option of *`Packer`*, `pack()` method doesn't return packed `bytes`.

You can use *`bytes()`* or *`getbuffer()`* to get packed data.

`bytes()` returns `bytes` object. `getbuffer()` returns some bytes-like object. It's concrete type is implement detail and it will be changed in future versions.

You can reduce temporary bytes object by using `Unpacker.getbuffer()`.

```
packer = Packer(use_bin_type=True, autoreset=False)

packer.pack([1, 2])
packer.pack([3, 4])

with open('data.bin', 'wb') as f:
    f.write(packer.getbuffer())

packer.reset()  # reset internal buffer
```

Python Module Index

Index